Day 2: Data structures and databases

Kenneth Benoit and Slava Mikhaylov

Introduction to Data Science and Big Data Analytics

16 August 2016

Day 2 Outline

- Big Data
  - Business intelligence
  - Architecture
- Data types and structures
- Dataset manipulation
- Compression
- Relational databases and SQL
- Data bases
  - Relational data bases
  - Normal forms

Big Data

From Wikipedia

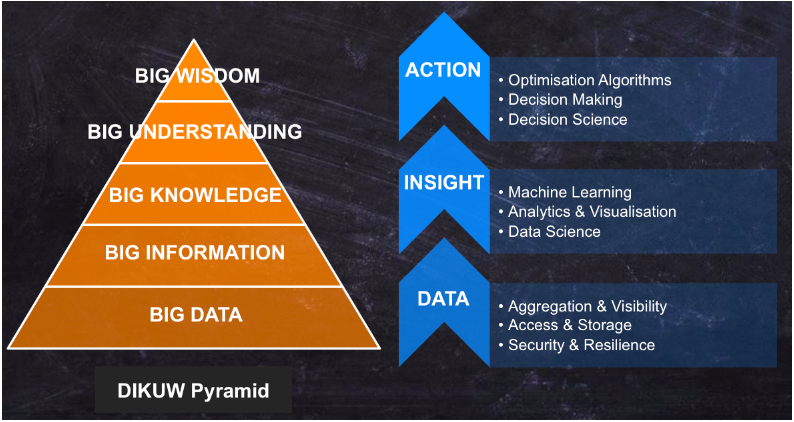
- Volume: big data doesn't sample; it just observes and tracks what happens
- Velocity: big data is often available in real-time
- Variety: big data draws from text, images, audio, video; plus it completes missing pieces through data fusion
- Machine Learning: big data often doesn't ask why and simply detects patterns
- Digital footprint: big data is often a cost-free byproduct of digital interaction

Components of Big Data

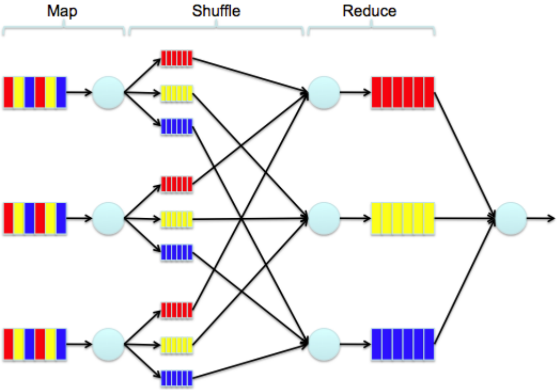
2011 McKinsey report "Big Data: The next frontier for innovation, competition, and productivity":

- Techniques for analyzing data (A/B testing, machine learning, natural language processing)
- Big Data technologies, like business intelligence, cloud computing and databases
- Visualization.

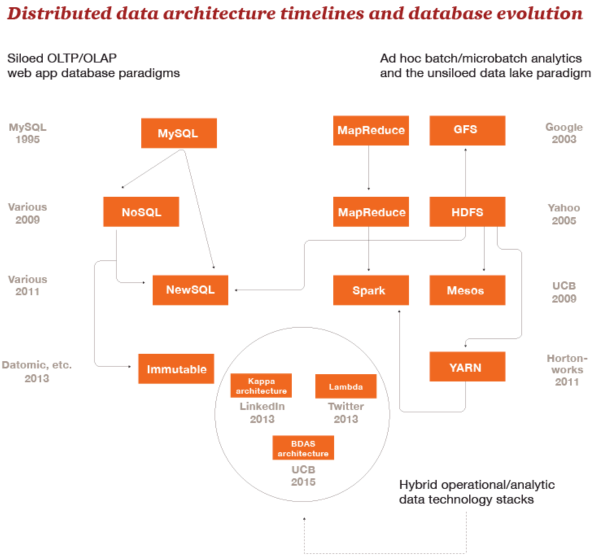
Business intelligence



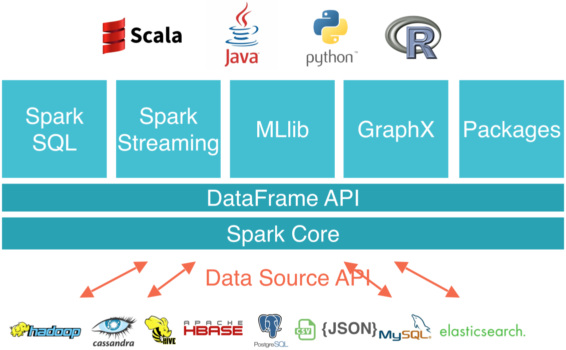
MapReduce



Architecture evolution



Typical data analytic stack



Big Data in R

- Two problems: RAM limitations for input data; computational time without natural parallizing.
- Three solutions:
  1. Scale-out vertically (get a bigger machine, e.g. AWS);
  2. Scale-out horizontally (change R behavior in not loading all data in memory and use distributed/parallel schema like MapReduce);
  3. Scale-out horizontally by interfacing with external distributed paradigms: Hadoop with `RHadoop`, HDFS with `rhdfs`; HBase with `rhbase`; `plyrmr` (big data version of `plyr`); a map-reduce API with `rmr2`; Spark with `SparkR`; MPI based `pbdr` for HPC.

Data types and structures

What is “data science”?

- extraction or generation of knowledge from data
- extends “data mining”, using computational and algorithmic methods
- combines applied statistical methods with advances in computer science, especially machine learning
- may involve “unstructured” data, especially text, but also video and images
- closely tied to computational methods

Why focus on data types and structures?

- "data" mining and "data" science imply that we know how to work with data
- data structures are not neutral - they shape how we record, see, and have the ability to analyze information
- much of the actual work in data mining and data analysis is done at the data "mungeing" stage

Basic (atomic) data types in R

numeric	8-byte numeric representations
integer	non-floating point numbers
character	text
logical	`TRUE` or `FALSE`

Recursive types also exist, such as lists and vectors; there are also special classifications for `NA`

Basic data types in R: integer

```
x <- 10
typeof(x)

## [1] "double"

is.integer(x)

## [1] FALSE

x <- 7L          # force integer type
typeof(x)

## [1] "integer"

object.size(x)

## 48 bytes

as.integer(3.14)

## [1] 3
```

Basic data types in R: character

```
typeof("test string")

## [1] "character"

object.size("a")

## 96 bytes

s <- ""              # Unicode
cat(s)

##

as.character("3.14")   # coerce numerics to character

## [1] "3.14"
```

Basic data types in R: numeric

```r
x <- 10.5          # assign a numeric value
x                  # print the value of x
```

```
## [1] 10.5
```

```r
typeof(x)           # print the class name of x
```

```
## [1] "double"
```

```r
object.size(x)     # show storage size in bytes
```

```
## 48 bytes
```

is.*() and as.*()

```
is.numeric(x)    # is the object of numeric type?

## [1] TRUE

is.numeric(7.1)

## [1] TRUE

is.numeric("7.1")

## [1] FALSE

is.numeric(as.numeric("7.1"))

## [1] TRUE
```

Basic data types in R: logical

A logical value is ‘TRUE‘ or ‘FALSE‘, often created via comparison between variables.

```
1 < 2              # is 1 less 2

## [1] TRUE

x <- c(1, 2, 3)
y <- c(4, 3, 2)
x > y              # vectorized comparison

## [1] FALSE FALSE  TRUE

typeof(x > y)

## [1] "logical"
```

Difference between ‘mode‘ and ‘class‘

- 'atomic' modes are numeric, complex, character and logical
- recursive objects have modes such as 'list' or 'function' or a few others
- an object has one and only one mode
- 'class' is a property assigned to an object that determines how generic functions operate with it - not a mutually exclusive classification
- an object has no specific class assigned to it, such as a simple numeric vector, it's class is usually the same as its mode, by convention
- an object's mode can be changed through coercion, without necessarily changing the class

Numerical precision issues

- floating point numbers are approximations of numbers
  - precision: anything more than 16 base-10 digits must be approximated
  - fractions: approximated if not $\frac{x}{2^k}$
  - anything over stated precision is truncated: $3.57\text{e}21 + 1 = 3.57\text{e}21$

```
1 - 4/5 - 1/5    # not zero!
```

```
## [1] -5.551115e-17
```

Machine limits

```
.Machine$integer.max
```

```
## [1] 2147483647
```

```
.Machine[c("double.xmin", "double.xmax", "double.digits")]
```

```
## $double.xmin
## [1] 2.225074e-308
##
## $double.xmax
## [1] 1.797693e+308
##
## $double.digits
## [1] 53
```

Alternatives (Stata)

- single and double precision: `http://blog.stata.com/2012/04/02/the-penultimate-guide-to-precision/`
- R has only double precision

Common input formats

- csv
- Excel
- "fixed formats"
- relational databases
- embedded tags: Extensible Markup Language (XML)
- key-value pair schemes (JSON)
  - examples of JSON and XML: `http://json.org/example`

Special issue: text encoding

- a "character set" is a list of character with associated numerical representations
- ASCII: the original character set, uses just 7 bits (2^7) – see `http://ergoemacs.org/emacs/unicode_basics.html`
- ASCII was later extended, e.g. ISO-8859 `http://www.ic.unicamp.br/~stolfi/EXPORT/www/ISO-8859-1-Encoding.html`, using 8 bits (2^8)
- but this became a jungle, with no standards: `http://en.wikipedia.org/wiki/Character_encoding`

Solution: Unicode

- Unicode was developed to provide a unique number (a "code point") to every known character – even some that are "unknown"
- problem: there are more far code points than fit into 8-bit encodings. Hence there are multiple ways to *encode* the Unicode code points
- *variable-byte* encodings use multiple bytes as needed. Advantage is efficiency, since most ASCII and simple extended character sets can use just one byte, and these were set in the Unicode standard to their ASCII and ISO-8859 equivalents
- two most common are UTF-8 and UTF-16, using 8 and 16 bits respectively

Warnings with text encodings

- Input texts can be very different
- Many text production software (e.g. MS Office-based products) still tend to use proprietary formats, such as Windows-1252
- Windows tends to use UTF-16, while Mac and other Unix-based platforms use UTF-8
- Your eyes can be deceiving: a client may display gibberish but the encoding might still be as intended
- No easy method of detecting encodings (except in HTML meta-data)

Dataset manipulation

What is a “Dataset”?

- A dataset is a “rectangular” formatted table of data in which all the values of the same variable must be in a single column
- Many of the datasets we use have been artificially reshaped in order to fulfill this criterion of rectangularity

Revisting basic data concepts

- The difference between tables and *datasets*

Revisting basic data concepts

- The difference between tables and *datasets*
- This is a (partial) dataset:

```
district     incumbf wonseatf
1       Carlow Kilkenny Challenger     Lost
2       Carlow Kilkenny Challenger     Lost
5       Carlow Kilkenny  Incumbent      Won
100 Donegal South West Challenger     Lost
459             Wicklow  Incumbent      Won
464             Wicklow Challenger     Lost
```

Revisting basic data concepts

- The difference between tables and *datasets*
- This is a (partial) dataset:

```
district     incumbf wonseatf
1       Carlow Kilkenny Challenger     Lost
2       Carlow Kilkenny Challenger     Lost
5       Carlow Kilkenny  Incumbent      Won
100 Donegal South West Challenger     Lost
459             Wicklow  Incumbent      Won
464             Wicklow Challenger     Lost
```

- This is a table:

```
           Lost Won
Challenger  266  60
Incumbent    32 106
```

Revisting basic data concepts

- The difference between tables and *datasets*
- This is a (partial) dataset:

```
district    incumbf wonseatf
1       Carlow Kilkenny Challenger     Lost
2       Carlow Kilkenny Challenger     Lost
5       Carlow Kilkenny  Incumbent      Won
100 Donegal South West Challenger     Lost
459             Wicklow  Incumbent      Won
464             Wicklow Challenger     Lost
```

- This is a table:

```
           Lost Won
Challenger  266  60
Incumbent    32 106
```

- The key with a dataset is that all the values of the same variable must be in a single column

Example: Comparative Manifesto Project dataset

Note: Available from `https://manifestoproject.wzb.eu/`

```
# load in a subset of the Manifesto Project dataset, with counts
load(url("http://kenbenoit.net/files/cmpdata.Rdata"))
# View(cmpdata)
```

Example: Comparative Manifesto Project dataset

This is “wide” format:

	country	countryname	oecdmember	eumember	edate	date	party	partyname
203	42	Austria	10	10	2008-09-28	200809	42320	SPOE Social Democratic Party
204	42	Austria	10	10	2008-09-28	200809	42110	Green Party
205	42	Austria	10	10	2008-09-28	200809	42520	OVP: People's Party
206	42	Austria	10	10	2008-09-28	200809	42420	FPO: Freedom Party
207	42	Austria	10	10	2008-09-28	200809	42710	BZO Alliance for the Future of Austria
208	42	Austria	10	10	2008-09-28	200809	42220	KP<d6> Communist Party of Austria
314	21	Belgium	10	10	1991-11-24	199111	21521	CVP Christian People's Party
315	21	Belgium	10	10	1991-11-24	199111	21111	ECOLO Francophone Ecologists
316	21	Belgium	10	10	1991-11-24	199111	21321	SP Flemish Socialist Party
317	21	Belgium	10	10	1991-11-24	199111	21522	PSC Christian Social Party
318	21	Belgium	10	10	1991-11-24	199111	21421	PVV Party of Liberty and Progress
319	21	Belgium	10	10	1991-11-24	199111	21913	VU People's Union
320	21	Belgium	10	10	1991-11-24	199111	21912	FDF Francophone Democratic Front

Long v. wide formats

- reshape
  - the “old” R way to do this, using ‘base::reshape()‘
  - problem: confusing and difficult to use
- reshape2
  - from Hadley Wickham’s `reshape2` package
  - data is first ‘melt‘ed into long format
  - then ‘cast‘ into desired format

Example: wide to long using reshape2

```
require(reshape2, quietly = TRUE)
# this will select only the "per" variables for measurement
cmpdataLong <- melt(cmpdata,
                    id.vars = c("countryname", "party", "date"),
                    measure.vars = names(cmpdata)[21:76],
                    variable.name = "category",
                    value.name = "catcount")
# why do this?
cmpdataLong$category <- as.character(cmpdataLong$category)
# View(cmpdataLong)
```

Example: wide to long using reshape2

```
require(reshape2, quietly = TRUE)
# now we can get summary statistics across countries, e.g. for economic
with(subset(cmpdataLong, grepl("^per7", category)),
     table(countryname, category))

##                   category
## countryname        per701 per702 per703 per704 per705 per706
##   Austria              34     34     34     34     34     34
##   Belgium              63     63     63     63     63     63
##   Cyprus               10     10     10     10     10     10
##   Denmark              60     60     60     60     60     60
##   Finland              47     47     47     47     47     47
##   France               23     23     23     23     23     23
##   Germany              30     30     30     30     30     30
##   Great Britain        20     20     20     20     20     20
##   Greece               17     17     17     17     17     17
##   Iceland              31     31     31     31     31     31
##   Ireland              31     31     31     31     31     31
##   Israel               32     32     32     32     32     32
##   Italy                41     41     41     41     41     41
##   Luxembourg           21     21     21     21     21     21
##   Malta                 4      4      4      4      4      4
##   Netherlands          48     48     48     48     48     48
##   Norway               28     28     28     28     28     28
##   Portugal             38     38     38     38     38     38
```

A better way

```
require(dplyr, warn.conflicts = FALSE, quietly = TRUE)
cmpdataLong2 <- melt(select(cmpdata, countryname, party, date, per101:per706),
                     id.vars = c("countryname", "party", "date"),
                     # NOT NEEDED measure.vars = names(cmpdata)[21:76],
                     variable.name = "category",
                     value.name = "catcount")
cmpdataLong2$category <- as.character(cmpdataLong2$category)
identical(cmpdataLong, cmpdataLong2)

## [1] TRUE
```

A better way

```
with(filter(cmpdataLong2, grepl("^per7", category)),
     table(countryname, category))

##                  category
## countryname       per701 per702 per703 per704 per705 per706
##   Austria             34     34     34     34     34     34
##   Belgium             63     63     63     63     63     63
##   Cyprus              10     10     10     10     10     10
##   Denmark             60     60     60     60     60     60
##   Finland             47     47     47     47     47     47
##   France              23     23     23     23     23     23
##   Germany             30     30     30     30     30     30
##   Great Britain       20     20     20     20     20     20
##   Greece              17     17     17     17     17     17
##   Iceland             31     31     31     31     31     31
##   Ireland             31     31     31     31     31     31
##   Israel              32     32     32     32     32     32
##   Italy               41     41     41     41     41     41
##   Luxembourg          21     21     21     21     21     21
##   Malta                4      4      4      4      4      4
##   Netherlands         48     48     48     48     48     48
##   Norway              28     28     28     28     28     28
##   Portugal            38     38     38     38     38     38
##   Spain               57     57     57     57     57     57
##   Sweden              44     44     44     44     44     44
```

Grouping operations: number of parties per election

```
# group by country-election
by_country <- group_by(cmpdataLong, countryname, date)
nparties <- summarise(by_country, npart = n())
head(nparties)

## Source: local data frame [6 x 3]
## Groups: countryname [1]
##
##   countryname   date npart
##         (chr)  (int) (int)
## 1     Austria 199010   224
## 2     Austria 199410   280
## 3     Austria 199512   280
## 4     Austria 199910   224
## 5     Austria 200211   280
## 6     Austria 200610   280

# is that correct?
```

Grouping operations: number of parties per election corrected

```
# group by country-election
by_country_unique <- distinct(cmpdataLong, countryname, date, party)
by_country_n <- group_by(by_country_unique, countryname, date)
nparties <- summarise(by_country_n, npart = n())
head(nparties, 10)
```

```
## Source: local data frame [10 x 3]
## Groups: countryname [2]
##
##    countryname   date npart
##          (chr)  (int) (int)
## 1      Austria 199010     4
## 2      Austria 199410     5
## 3      Austria 199512     5
## 4      Austria 199910     4
## 5      Austria 200211     5
## 6      Austria 200610     5
## 7      Austria 200809     6
## 8      Belgium 199111    11
## 9      Belgium 199505    10
## 10     Belgium 199906     9
```

Grouping operations: number of parties per election final

```r
# using "chaining" -- no need for intermediate objects
nparties2 <- distinct(cmpdataLong, countryname, date, party) %>%
    group_by(countryname, date) %>%
    summarise(npart = n())
identical(nparties, nparties2)

## [1] TRUE
```

Compression in general

- Seeks to economize on space by representing recurring items using patterns that represent the uncompressed data
- Common in formats for graphics and video encoding
- "Lossless" formats compress data without reducing information - examples are .zip and .gz compression
- This (and avoiding errors) is also a principle in normalized relational data forms
- Also very important for sparse matrix representations, where many of the cells are zero, but it would be very wasteful to record a double precision numeric zero for each of these non-informative cells

Compression: sparse matrix format

Compression: sparse matrix format

used because many forms of matrix are very sparse - for example, document-term matrixes

```
require(quanteda, warn.conflicts = FALSE, quietly = TRUE)

## quanteda version 0.9.7.22

myDfm <- dfm(inaugTexts, verbose = FALSE)
myDfm[1:10, 1:5]

## Document-feature matrix of: 10 documents, 5 features.
## 10 x 5 sparse Matrix of class "dfmSparse"
##                  features
## docs               fellow-citizens  of the senate and
##   1789-Washington                1  71 116      1  48
##   1793-Washington                0  11  13      0   2
##   1797-Adams                     3 140 163      1 130
##   1801-Jefferson                 2 104 130      0  81
##   1805-Jefferson                 0 101 143      0  93
##   1809-Madison                   1  69 104      0  43
##   1813-Madison                   1  65 100      0  44
##   1817-Monroe                    5 164 275      0 122
##   1821-Monroe                    1 197 360      0 141
##   1825-Adams                     0 245 304      0 116
```

Compression: sparse matrix format

used because many forms of matrix are very sparse - for example, document-term matrixes

```
# how many cell counts are zeros
sum(myDfm==0) / (ndoc(myDfm) * nfeature(myDfm)) * 100

## [1] 91.67661

object.size(myDfm)

## 1125120 bytes

object.size(as.matrix(myDfm))

## 4762560 bytes
```

Sparse matrix formats

- "simple triplet" format

  i indexes row
  j indexes column
  x indicates value

- "compressed sparse column" format

  i indexes row
  p indexes the first nonzero element in each column of the matrix
  x indicates value

Sparse matrix format examples

```r
# create a sparse matrix
require(Matrix)

## Loading required package:  Matrix

spTmatrix <- as(Matrix(c(0, 0, 1, 2, 4, 0, 3, 0, 5), nrow = 3), "dgTMatrix")
spTmatrix

## 3 x 3 sparse Matrix of class "dgTMatrix"
##
## [1,] . 2 3
## [2,] . 4 .
## [3,] 1 . 5
```

Simple triplet matrix

```
spTmatrix

## 3 x 3 sparse Matrix of class "dgTMatrix"
##
## [1,] . 2 3
## [2,] . 4 .
## [3,] 1 . 5

str(spTmatrix)

## Formal class 'dgTMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:5] 2 0 1 0 2
##   ..@ j       : int [1:5] 0 1 1 2 2
##   ..@ Dim     : int [1:2] 3 3
##   ..@ Dimnames:List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##   ..@ x       : num [1:5] 1 2 4 3 5
##   ..@ factors : list()
```

Compressed sparse column matrix

(default for 'Matrix()' – defaults to *column* rather than *row* compressed format because R uses "column major" order matrixes)

```
spCmatrix <- as(spTmatrix, "dgCMatrix")
str(spCmatrix)

## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:5] 2 0 1 0 2
##   ..@ p       : int [1:4] 0 1 3 5
##   ..@ Dim     : int [1:2] 3 3
##   ..@ Dimnames:List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##   ..@ x       : num [1:5] 1 2 4 3 5
##   ..@ factors : list()
```

Compressed sparse column matrix

(default for ‘Matrix()‘ – defaults to *column* rather than *row* compressed format because R uses ”column major” order matrixes)

```
spCmatrix <- as(spTmatrix, "dgCMatrix")
str(spCmatrix)

## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i        : int [1:5] 2 0 1 0 2
##   ..@ p        : int [1:4] 0 1 3 5
##   ..@ Dim      : int [1:2] 3 3
##   ..@ Dimnames:List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##   ..@ x        : num [1:5] 1 2 4 3 5
##   ..@ factors : list()
```

Transactional databases

- Transactional databases are "ACID"
  - **A** tomic all of a transaction is completed, or none is completed
  - **C** onsistent all completed transactions leave DB in a state compliant with rules
  - **I** solated no results are visible until transaction is completed
  - **D** urable completed transactions persist, even if program crashes or exits
- "SQL" – sttructured query language – database manaers are the most common type of relational database manager. Many variations exist, including Oracle, MySQL, PostgreSQL, and SQLite
- Most of these systems use slight variations on the same syntax, i.e. SQL
- Non-SQL based alternatives exist that do not require fixed schemas, in terms of table structures, data types, etc.
- "compressed sparse column" format

Databases

Relational data bases

- invented by E. F. Codd at IBM in 1970
- A relational database is a collection of data organized as a set of formally defined tables
- These tables can be accessed or reassembled in many different ways without having to reorganize the underlying tables that organize the data
- RDBMS: a relational database management system. Examples include: MySQL, SQLite, PostgreSQL, Oracle. MS Access is a lite version of this too.
- The standard user and application programmer interface to a relational database is structured query language (SQL)

Example

- example: Database of Parties, Elections, and Governments (DPEG) relational database

```
SELECT c.countryName, c.countryAbbrev, p.* FROM party AS p
    LEFT JOIN country AS c
    ON p.countryID = c.countryID
```

- simpler example: convert CMP data into relational tables for countries, parties, elections, categories, and counts

Basic relational structures

- tables
  - also known as "relations"
  - tables define the forms of the data that are linked to other data through key relations
- keys: how tables are cross-referenced
  - primary key: an column in a table that uniquely identifies the remaining data in the table
  - foreign key: a field in a relational table that matches the primary key column of another table
  - `join` operations link tables in a structured query

Normal forms 1

"Normalizing" a database means creating a proper set of relations
First normal form: No Repeating Elements or Groups of Elements

```
head(select(cmpdata, countryname, partyname, date, per108, per110))

##     countryname                        partyname   date per108 per110
## 175     Austria              FP Freedom Party 199010      3      0
## 176     Austria          GA Green Alternative 199010      0      3
## 177     Austria SP Social Democratic Party 199010      5      0
## 178     Austria            VP People's Party 199010      8      0
## 179     Austria              FP Freedom Party 199410      1     11
## 180     Austria             LF Liberal Forum 199410      0      0
```

Here, this is violated because of the wide format of per108 and per110. To solve this, we have to move this to long format.

Normal forms 2

Second normal form: No Partial Dependencies on a Concatenated Key

```
head(cmpdataLong)

##   countryname party   date category catcount
## 1     Austria 42420 199010   per101        0
## 2     Austria 42110 199010   per101        0
## 3     Austria 42320 199010   per101        0
## 4     Austria 42520 199010   per101        5
## 5     Austria 42420 199410   per101        0
## 6     Austria 42421 199410   per101        0
```

Here, the format is still violated, because party 42420 is repeated. To solve this we need to create a party table and link to it using a foreign key.

Normal forms 3

Third normal form: No Dependencies on Non-Key Attributes. Every non-prime attribute of data in a table must be dependent on a primary key.

```
head(cmpdataLong)
```

```
##   countryname party   date category catcount
## 1     Austria 42420 199010   per101        0
## 2     Austria 42110 199010   per101        0
## 3     Austria 42320 199010   per101        0
## 4     Austria 42520 199010   per101        5
## 5     Austria 42420 199410   per101        0
## 6     Austria 42421 199410   per101        0
```

Here, this is violated because election data repeats across multiple values of the category count table, when it should have its own table.

Non-relational data

- recently popularized because so much data is unstructured, and dealing with new data forms in a classic relational setting requires changing the entire schema
- non-relational systems typically define data using key-value pairs
- example: JSON - see `http://kenbenoit.net/files/JSONexample.json`